All you need to know about the Smart Soccer Ball

Theodore Stone, Nathaniel Stone,
Xiang Guan, Srihari Nelakuditi
Computer Science and Engineering
University of South Carolina, Columbia

1. INTRODUCTION

The Adidas miCoach Smart Ball is a size 5 regulation weight soccer ball containing sensors capable of measuring impact forces on the ball [1]. It is marketed as a means to improve one's kicking prowess. The provided miCoach app[1] for interacting with the ball is only suitable for its intended purpose of training players, providing only rudimentary speed and spin data from single, stationary kicks. All details regarding the inner workings and functions of the ball are proprietary, making the development of custom applications for the smart ball difficult. This paper details the insights and operation of the ball we have gleaned from the reverse engineering process.

2. BLE OVERVIEW

The Smart Ball communicates via Bluetooth Low Energy[2] (BLE). It acts as a server (BLE host) in connections. The Smart Ball broadcasts standard BLE services as well as custom services specific to its functionality.
"Compared to Classic Bluetooth, Bluetooth Smart is intended to provide considerably reduced power consumption and cost while maintaining a similar communication range [3]."
Presented below is a summary of key terms and concepts concerning BLE, copied from the Android Developer page on the subject [2]:

- **Generic Attribute Profile (GATT) -** The GATT profile is a general specification for sending and receiving short pieces of data known as "attributes" over a BLE link. All current Low Energy application profiles are based on GATT.
  - The Bluetooth SIG defines many profiles for Low Energy devices. A profile is a specification for how a device works in a particular application. Note that a device can implement more than one profile. For example, a device could contain a heart rate monitor and a battery level detector.
- **Attribute Protocol (ATT) -** GATT is built on top of the Attribute Protocol (ATT). This is also referred to as GATT/ATT. ATT is optimized to run on BLE devices. To this end, it uses as few bytes as possible. Each attribute is uniquely identified by a Universally Unique Identifier (UUID), which is a standardized 128-bit format for a string ID used to uniquely identify information. The attributes transported by ATT are formatted as characteristics and services.
- **Characteristic -** A characteristic contains a single value and 0-n descriptors that describe the characteristic's value. A characteristic can be thought of as a type, analogous to a class.
- **Descriptor -** Descriptors are defined attributes that describe a characteristic value. For example, a descriptor might specify a human-readable description, an acceptable range for a characteristic's value, or a unit of measure that is specific to a characteristic's value.
- **Service -** A service is a collection of characteristics. For example, one could have a service called "Heart Rate Monitor" that includes characteristics such as "heart rate measurement." A list of existing GATT-based profiles and services can be found on bluetooth.org.

Figure 1 contains a comprehensive list of GATT services and characteristics broadcasted by the smart ball.

3. HARDWARE OVERVIEW

The microcontroller used is the MSP430F5328, built by Texas Instruments. The accelerometer chip used is the LSM303[3], built by STMicroelectronics. The Bluetooth chip used is the nRF8001, built by Nordic Semiconductor.
The identities of the specific hardware chips used in the interior of the ball were determined in a variety of ways:

- Images of the interior of the ball showing the hardware
- Third party reports [6]
- Human readable references in debugging strings visible in the compiled firmware file[4]

The following three sections describe the various hardware components of the ball.

3.1 MSP430F5328

The microcontroller used in the ball is the MSP430F5328, built by Texas Instruments around a 16 bit CPU. It is designed for low cost and low power consumption. The MSP430 does not have an external memory bus, so it is limited to on-chip memory, and it is difficult to move data off the chip due to the lack of a DMA output strobe.

[1] Referred to as the companion app throughout this document
[2] Marketed as Bluetooth Smart
[3] The exact model number is still not known
[4] These strings were visible in the binary firmware file; no decompilation was necessary to see them. Some strings were obviously debugging messages and included references to the actual chip names used in the ball.

Service		Characteristic			
UUID	Name	UUID	Mnemonic	Default Value	Access
1800	Generic Access	2A00	Device Name	DB00XXXXX	R/W[1]
		2A01	Appearance	911	R
		2A04	Peripheral Preferred Connection Parameters	13107206[2]	R
1801	Generic Attribute	2A05	Service Changed	0	I/R
180A	Device Information	2A23	System ID	various	R
		2A24	Model Number String	nrfXXXXX	R
		2A25	Serial Number	various	R
		2A26	Firmware Revision String	3.5 (currently)	R/W
		2A28	Software Revision String	Rev1.0	R
		2A29	Manufacturer Name String	Adidas AIT	R
180F	Battery Service	2A19	Battery Level	varies	R
AD04 Smart Ball Service		AD13	???	0	N/R
		AD14	Logging Characteristic	0	N/R
		AD15	Command Characteristic	N/A	WNR
		AD16	Firmware Write Field	N/A	W
		AD17	Response Buffer Characteristic	N/A	N
		AD20	Sample Rate Characteristic	[1,0]	N/R
		AD1F	Status Characteristic	varies	R
		ADFE	Error Characteristic	varies	N/R
		AD12	Kick Event Characteristic	varies	N/R
		AD33	File Size Characteristic	varies	R
		DB55	???	N/A	N

[1] Even though this characteristic has write access and can be edited, it is reset to its default value automatically every time the ball is charged.
[1] This value signifies: min connection interval = 6, max connection interval = 200, slave latency = 0,

Figure 1: The GATT services and characteristics broadcasted by the ball

3.2 LSM303

The LSM303DLCH is a sensor chip built by STMicroelectronics containing a 3D digital linear acceleration sensor, a 3D digital magnetic sensor, and a thermometer.
Two operating power modes are available, normal and low power. While normal mode guarantees high resolution, low-power reduces the current consumption of the chip [4]. While we do not know which mode is used by the ball, we are guessing that low power mode is used, as this is in line with the use of the MSP430 and Bluetooth Low Energy, both of which emphasize low power consumption.
The output data rate and the scale / precision of the measured acceleration of the sensor can be selected from several options. This selection is done by the manufacturer, and it is likely that the firmware on the microcontroller has the selection code. It may be possible to edit these selections wirelessly but this is currently unknown.
All acceleration values are represented as signed integers of 12 bits left-aligned in a 16 bit space [4].

3.3 nRF8001

The Bluetooth chip used in the smart ball is the nRF8001, built by Nordic Semiconductor. The nRF8001 emphasizes ultra-low power consumption. It is a fully qualified Bluetooth Smart v4.0 Connectivity IC with integrated Radio, Link Layer, and Host stack supporting Peripheral (Slave) role operation [5].

3.4 Hardware Mounting

Each of the above three components are on a single board suspended within the interior of the ball. The board is enclosed in a plastic sphere of about 1.5 inches in diameter. The sphere is connected to the ball by 12 rubberized bands, distributed evenly around the surface of the sphere in the same configuration as the faces on a regular dodecahedron2. In addition to these 12 connections, there is a power cable that connects the board to the induction charging coils on the interior of the ball's surface[5].

Figure 2: (a) Ball cross-section; (b) Regular dodecahedron.

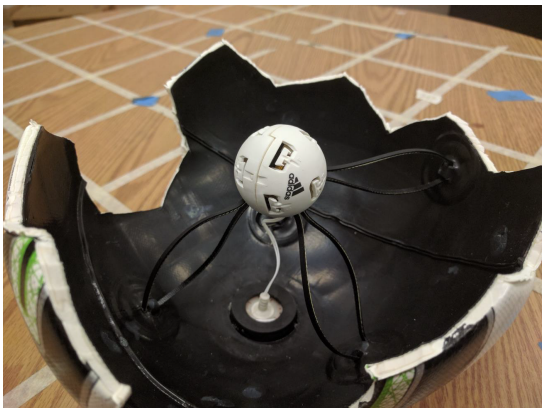

Figure 3: The sphere with several bands removed. The charging cable is visible extending beneath the sphere. The sphere was fused together as two halves; the seam can be seen above.

4. DETERMINING SOFTWARE PROTOCOL

As all information concerning the ball's functionality is proprietary, the protocol for communicating with and receiving data from it had to be reversed engineered.
The communication protocol was obtained by monitoring all messages sent between a ball and its companion app. To accomplish this, two separate applications were developed; one to emulate the companion app and another to emulate a smart ball. Both apps were developed for Android, using standard BLE libraries.
The companion app emulator was developed to simply connect to a real smart ball and report all messages sent from it.
The smart ball emulator was designed to broadcast the same signals emitted by the physical ball and connect to the real companion app and report all messages from it.
Both apps are capable of both receiving and sending commands

[5]This cable is significantly longer than the rubberized bands so as not to interfere with their connection

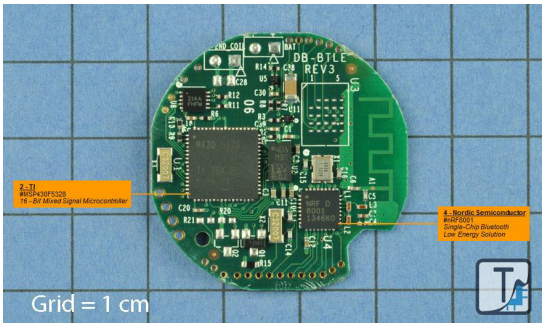

Figure 4: The board in the center of the sphere. Note that the accelerometer is positioned in the center of the board.

to and from their respective target applications, allowing the complete communication protocol to be eavesdropped. Figure 5 reiterates this concept.

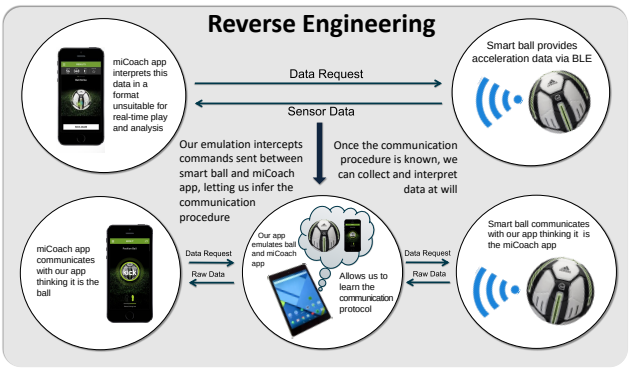

Figure 5: Reverse engineering the ball's protocol

The following two sections cover the protocols for the primary functions of the ball, recording and receiving data. The following notation is used:

- GATT characteristics are all referred to by their UUIDs. All BLE GATT UUIDs are of the form *0000XXXX-0000-8000-00805F9B34FB*; for simplicity only *XXXX* is shown. All characteristics shown belong to service *AD04*.
- Commands sent from the ball to the app are denoted by B $\Longrightarrow$ A while those sent from the app to the ball are marked A $\Longrightarrow$ B.
- [X,X,...] Represents an array of signed, 8 bit integers.

4.1 Capture Impact Protocol

This protocol prepares the smart ball to record a 1 second burst of accelerometer data. The ball is first put in a âĂŸreadyâĂŹ state where it waits for an impact. When an impact is detected by the accelerometer[6], it records and saves the data around the impact to an internal buffer. The data can be read from this buffer with a later command. Notifications concerning the state of the ball are sent to the client throughout the sequence of events.

[6]By detecting some threshold change in acceleration

1. A $\Longrightarrow$ B : Preparation command
   - Write [6] to characteristic AD15
2. B $\Longrightarrow$ A : Success callback
   - Notify [250] to characteristic ADFE
3. A $\Longrightarrow$ B : Clear ball's internal buffer
   - Write [3] to characteristic AD15
4. B $\Longrightarrow$ A : Ball signals it is waiting for impact
   - Notify [1] to characteristic AD1F
5. B $\Longrightarrow$ A : Ball signals impact has occurred
   - Notify [0] to characteristic AD1F

4.2 Request Data Protocol

This protocol requests the data stored in the smart ballâĂŹs internal buffer from it's last recorded impact. Between 0 and 1096[7] samples of data can be requested, specified as a 16 bit, little endian integer (S_L and S_H in the command, were S_L is the low byte and S_H is the high byte). Additionally, data can be formatted in three different ways, specified by T. T = 0 will result in loosely packed data with one sample per line, T = 1 corresponds to tightly packed data with multiple samples per line, and T = 2 will result in the data coming in a compressed format. Other values of T will result in no data. The same data can be requested multiple times in any of the three formats. See section 5 for descriptions on all three data formats.

1. A $\Longrightarrow$ B : Data request command
   - Write [10, 0, 0, 0, 0, S_L, S_H, 0, 0, T] to characteristic AD15
2. B $\Longrightarrow$ A : Data from ball
   - Notify [X,X,...,X] to characteristic AD17[8]

5. DATA TYPE COMPARISON

Section 4 describes how to retrieve data from the soccer ball. This data can be requested in three different formats, which will be referred as types 0, 1, and 2[9] throughout this document.
Type 0 and type 1 data represent the acceleration data in its native format as signed 16 bit integers, while type 2 data is a compressed version of the data.
The acceleration samples given by the ball are represented as signed (2's complement) 16 bit integers[10], with maximum and minimum values of 2040, and -2039, respectively (corresponding to ±4 g's, with a sensitivity of 0.002 g / LSB[11]).
When the smart ball experiences and records an impact, the acceleration data is stored in an internal buffer[12]. This data can

[7]Higher numbers can be specified, but all data past 1096 samples will be 0
[8]Each Notify contains 20 bytes of data and Notifies continue until all data has been sent.
[9]These numbers represent the value of the T field mentioned in section 4.2
[10]Although only the right most 12 bits are ever used
[11]Least Significant Bit
[12]Probably about 6.5 KB; the maximum number of samples that can be requested from the ball is 1096, each sample consists of x, y, and z values each of which are represented by 16 bits

then be requested by the user, and is transmitted in one of the three available formats by specifying a byte in the data request code.

5.1 Type 0 Data

Type 0 data begins with the data request code, preceded by -118 and padded with trailing zeros to be 20 bytes long:

[-118, 10, 0, 0, 0, 0, 72, 412, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Each successive sequence of 14 bytes corresponds to one sample containing an x, y, and z acceleration value preceded by a 2 byte sequence number. Each x, y, and z sample is 2 bytes long and is formatted as a little endian value. The last 6 bytes are occupied by padding zeros.

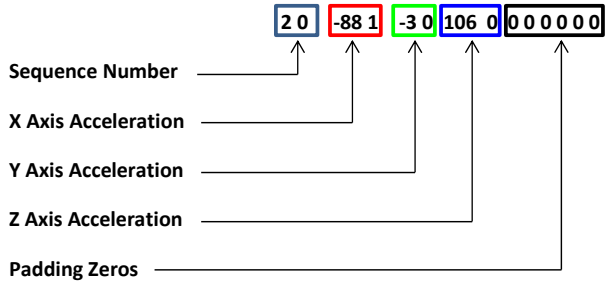

Figure 6: Type 0 data line format

Type 0 data ends with the following line:

[-102, -25, 0, 0, 0, 74, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

5.2 Type 1 Data

Type 1 data begins with two lines; one is the data request code preceded by -118 and padded with trailing zeros to be 20 bytes long, the other is a line of 20 zero bytes:

[-118, 10, 0, 0, 0, 0, 72, 4, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Each successive sequence of 20 bytes corresponds to three samples containing x, y, and z acceleration values preceded by a 2 byte sequence number. Each x, y, and z component is 2 bytes long and is formatted as a little endian value. Unlike type 0 data, type 1 data completely uses the available 20 bytes per line.

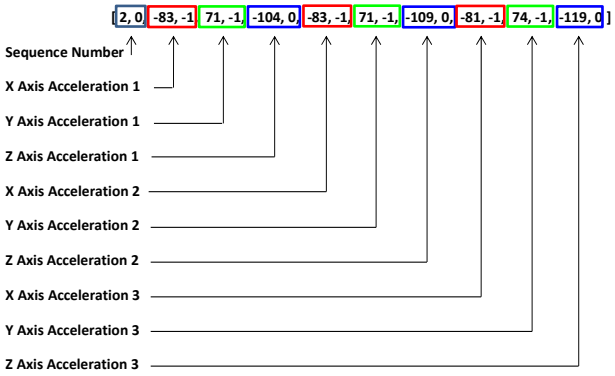

Figure 7: Single line of type 1 data

Type 1 data ends with the following line:

[-102, 110, 1, 0, 0, 73, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

5.3 Type 2 Data

Type 2 data is a compressed version of the type 1 and type 0 data. The compression algorithm is very simple and reduces the size of most data files roughly 20 to 30 percent.
The compression works as follows:
Each type 2 data file is made up of 20 byte lines. Each line is of the following form:

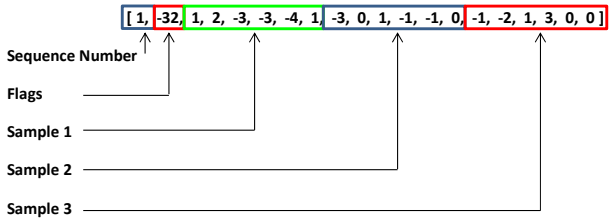

Figure 8: Single line of type 2 data.

Each of the three samples contains 6 bytes and can either be 'compressed' or 'uncompressed,' explained shortly.
The **Flags** field is of the form XXX0 0000 where the three high order bits are uses to indicate whether their corresponding sample (sample 1, 2, or 3 in the figure above, where the first X corresponds to sample 1) is compressed (1) or not (0).
An uncompressed sample has exactly the same form as a sample in type 1 and type 2 data, with three signed, 16 bit little endian components corresponding to the x, y, and z axes.
A compressed sample is a little different; it actually contains two samples specified as offsets of the previous sample. For instance, if the previous sample has component values X, Y, and Z, then a compressed sample corresponds to two uncompressed samples with components X + b0, Y + b1, Z + b2, and X + b3, Y + b4, Z + b5, where b0 to b5 are the 6 bytes in the compressed sample.
This scheme takes advantage of the fact that many samples will be similar in value to samples near them and therefore donâĂŹt need to be expressed with two full bytes per component.
Like the other types of data, type 2 data begins and ends with specific lines. The first line is the data request code preceded by -118 and followed by padding zeros.
Type 2 data begins with:

[-118, 10, 0, 0, 0, 0, 72, 4, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0, 0, 0]

Type 2 data ends with:

[-102, -62[13], 0, 0, 0, 74, 4, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

This algorithm was inferred from the decompiled Android version of the Adidas miCoach app.
NOTE: The first line of data from the smart ball is always uncompressed, and the miCoach Smart Ball app will reject any data that does not adhere to this rule.

6. REVERSE ENGINEERING HARDWARE SETTINGS

Section 3 describes the different hardware components used in the Smart Ball but does not mention the specific settings used in the ball's configuration. This section will cover the specific settings that the ball's hardware employs and the procedures used to infer them.

[13]This byte is the sequence number continued from the previous line and so can take other values. All other bytes are fixed.

6.1 Sample Rate of Accelerometer

The frequency of the accelerometer inside the Smart Ball is 1 KHz, meaning that acceleration samples are captured and recorded every millisecond. Interestingly, the LSM303DLHC does not have an Output Data Rate (ODR) setting for 1 KHz. This could indicate that the microcontroller does some processing on the data.
The sample rate was found by recording data for repeated bounces of the ball from both the internal accelerometer and a microphone. Peaks appear in both data sets that correspond to the ball bouncing, see figure 9. From the accelerometer data, the number of samples between peaks corresponds to an amount of time between peaks in the audio data. Dividing one by the other gives the number of samples per second, which is the sample rate.

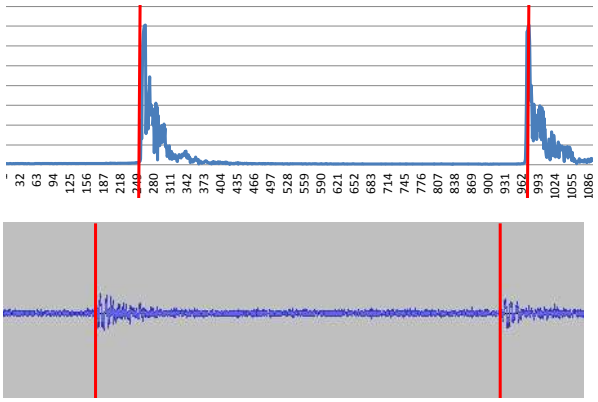

Figure 9: Acceleration data (top) and corresponding audio data (bottom) with two bounces visible.

6.2 Accelerometer Scale and Precision

The LSM303 chip has four different configurations, seen in figure 10. The Smart Ball uses the ±4 g range with 0.002 mg / LSB[14] precision setting.
Accelerometer values are represented by signed (2's complement) 16 bit integers. The actual raw format given by the accelerometer in the ball is a 12 bit signed integer left-aligned in a 16 bit space, meaning only 12 bits convey meaningful information. All raw values in this report will be in the 12 bit form, with the unused right bits shifted out. Due to the limited width of the data value representation, higher measurement ranges correspond to lower measurement precisions (measured in mg / LSB).

Available Accelerometer Configurations				
Measurement Range	±2 g	±4 g	±8 g	±16 g
Measurement Sensitivity[3]	1 mg/LSB	2 mg/LSB	4 mg/LSB	12 mg/LSB

Figure 10: Available accelerometer configurations for the LSM303DLHC

The procedure to determine these settings was straight forward. At rest, the ball should experience exactly 1 g of acceleration while it experiences 0 g's in free fall. The intuitive explanation is this: When stationary, a force is being exerted on the sensor to keep it from falling (in this case the rubberized bands in the ball). This force corresponds to 1 g of acceleration. When falling, this force is no longer exerted, as the sensor is falling freely and uninhibited. Therefore, in this state, 0 g's are registered by the sensor. The correct configuration can be found by simply determining which setting resolves stationary accelerometer readings to 1 g. Furthermore, after determining the sensitivity settings used by the ball, the minimum and maximum measured acceleration values (as signed 16 bit integers) should correspond to the maximum measurable values of the chip given the sensitivity setting (after being converted to g's by being multiplied by sensitivity).
The complete process is as follows:

1. Find the scale and sensitivity that convert stationary accelerometer readings to 1 g.
2. Verify that the minimum and maximum measured values evaluate to the acceleration range corresponding to the selected sensitivity setting when multiplied by the sensitivity.

The accelerometer readings for a stationary ball can be collected by examining the leading values from an impact where the ball was first stationary before being hit or kicked.
The minimum and maximum acceleration values can be determined by examining the maximum and minimum values from many impact graphs.

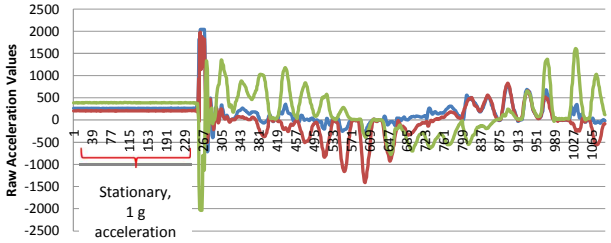

Figure 11: 1 g of acceleration before impact

The stationary acceleration as a signed 12 bit integer was found to be around 511. This value corresponds to 1 g only with the ±4 g, 2 mg/LSB range and sensitivity setting[15]. Furthermore, the maximum and minimum raw values outputted by the accelerometer are 2040 and -2039, respectively, which, with a sensitivity of 2 mg/LSB, correspond to accelerations of 4.08 and -4.078 g's, which match the required ±4 g's corresponding to the 2 mg/LSB sensitivity.

Stationary Acceleration Magnitude	~ 511
Maximum Measured Acceleration	2040
Minimum Measured Acceleration	-2039

Figure 12: Measured raw accelerometer values

To verify these findings, an accelerometer of known settings was attached to the Smart Ball's accelerometer and both were subjected to a series of accelerations. The known accelerometer has a maximum range of 200 g's, a precision of 1/163.8 g/LSB, and a sample rate of 800 Hz. Figure 13 shows the results of this test.

7. ADAPTING THE SMART BALL FOR LIVE DATA COLLECTION

One of the most attractive applications for the Smart Ball is its use an in-game sensor to record impacts occurring in a live soccer match. This adaptation, while theoretically sound, is complicated by the inherent hardware and software limitations of the ball. These limitations can be summarized as follows:

[14] Least Significant Bit

[15] As $511 \times 0.002 = 1.022 \approx 1$ g

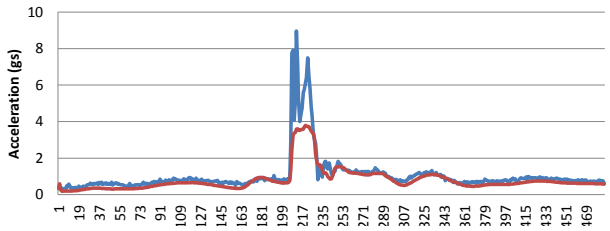

Figure 13: External accelerometer (blue) paired with the Smart Ball's sensor (red). Notice the peak truncation on the Smart Ball's sensor due to the limited ± 4 g range.

1. *Hardware* - The hardware used in the Smart Ball is optimized for its intended environment and application, which focuses on collecting individual, brief impacts from an initially stationary ball, and emphasizes long battery life and therefore low power consumption. These expectations translate to limited measurement range[16], weak transmission signal, and low onboard memory.

2. *Software* - The software is also targeted to solely support the intended application of the ball. This requirement is manifested in the data collection and retrieval protocol, which prevents more than one impact from being stored at a time between transmissions.

Dealing with these limitations is the primary focus while adapting the Smart Ball for live data collection.

7.1 Collecting Data Continuously

Three general approaches can be used to collect continuous data from a sensor:

1. Record and write data to an internal buffer for offline transmission

2. Simultaneously record and transmit data

3. A hybrid of the two previous approaches where data is recorded and then transmitted

The communication protocol and the limited memory of the ball prevents the use of the first approach, as only a single second-long impact may be stored in the Smart Ball. The second approach is also impossible, because the Smart Ball cannot simultaneously record and transmit data. Therefore the third approach must be taken.
Because simultaneous recording and transmission is not possible, a truly continuous stream of data cannot be retrieved from the ball. Instead this stream must be simulated by continuously cycling through the data collection and retrieval protocols of the Smart Ball. A simplified formula for the latency required for one such iteration is given below:

$$T = T_{wait\_for\_impact} + T_{record} + T_{transmit}$$

where $T_{wait\_for\_impact}$ is the time for the ball to detect an impact to record, T_{record} is the time to record the impact, and $T_{transmit}$ is the time to transmit the recorded data. The times required to send the intermediate commands as required by the communication protocol are negligible and are therefore ignored.
In practice the latter component $T_{transmit}$ is the bottleneck, often lasting 30 seconds to more than a minute. This is due to the low energy Bluetooth hardware used in the ball. This large transmission delay results in an effective coverage of less than 2 percent at best, obviating the use of the Smart Ball in its off the shelf configuration for continuous sensing.

[16]The intended application has a relatively highly controlled environment where the ball must be stationary prior to impact, allowing for less measurement ability in the accelerometer. This limitation is also compensated for in the companion app by the sophistication of a regression model

7.2 Collecting Discrete Events

The most effective means of utilizing the Smart Ball for data collection in practice has been to target discrete events in a game. These events are often chosen so that the probability of an ensuing target impact is relatively large.
This method requires one spectator who must trigger the ball before a key impact, such as when the ball is kicked high in the air. The spectator must have sufficient time to notify the ball. The downside of this approach is that such situations when there is a large enough amount of warning are rare in a game, and, when they do occur, an ensuing event of interest is not guaranteed.

7.3 Conclusion

While both of the approaches presented in this section suffer from severe limitations, the latter approach has proved most useful in practice.

8. REFERENCES

[1] Adidas miCoach: The Interactive Personal Coaching and Training System. http://micoach.adidas.com/smartball/.

[2] Android BLE. http://developer.android.com/guide/topics/connectivity/bluetooth-le.html.

[3] Blow Tooth Low Energy. https://en.wikipedia.org/wiki/Bluetooth_low_energy.

[4] LSM303DLHC: Ultra compact high performance e-compass 3D accelerometer and 3D magnetometer module. https://www.pololu.com/file/download/LSM303DLHC.pdf?file_id=0J564.

[5] Nordic Semiconductor nRF8001. https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF8001.

[6] World Cup Teardown. http://www.techinsights.com/teardown.com/world-cup-teardown.